Applications of Dual Quaternions in Three Dimensional Transformation and Interpolation

November 11, 2013

Matthew Smith
mrs126@uclive.ac.nz
Department of Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand

Supervisor: Dr R. Mukundan
mukundan@canterbury.ac.nz

Abstract

Quaternions have long been integral to the field of computer graphics, due to their minimal and robust representation of rotations in three dimensional space. Dual quaternions represent a compact method of representing rigid body transformations (that is rotations and translations) with similar interpolation and combination properties. By comparing them to two other kinds of rigid transformations, we examine their properties and evaluate their usefulness in a real time environment. These properties include accuracy of operations, efficiency of operations, and the paths that interpolation and blending methods using those transformation methods take. The blending and interpolation methods are of particular interest as we constructed a skeletal animation system to highlight a potential application of dual quaternions. The bone hierarchy was constructed with dual quaternions and a sequence of identical hierarchies with different transformations at each bone can be interpolated as though they were keyframes to produce animations. Weighted transformations required in skinning the skeleton structure to a triangular mesh also prove an effective application of dual quaternions. Our findings show that while dual quaternions are useful in the context of skeletal animation, other applications may favour other representations, due to simplicity or speed.

Contents

1 Introduction

Rigid transformations are transformations capable of representing rotations, reflections and translations, with proper rigid transformations represented by rotations and translations[22]. Mathematically they are any vector space map that preserves the distances between all pairs of points. Rigid transformations of $\mathbb{R}^3$ can be considered as all the possible transformations or orientations of a rigid body within 3D space. Thus the primary application of rigid transformations is the representation of the position and orientation of rigid bodies. Another use is their ability to represent changes in coordinate systems that are similarly distance preserving.

Rigid transformations[1] as they are formalised, only exist as a means of transforming a vector space. However when considering a representation of vector spaces, specifically in software as a sequence of real numbers, one needs an appropriate representation for rigid transformations. It turns out there are a number of ways to represent these, but the representation that is chosen is important[11][14]. While all representations should transform points and chain with other transformations identically, different representations will change the accuracy and efficiency of performing these operations in a software implementation. Different representations may also change how efficient and simple its was to perform other operations such as interpolation[12].

Matrices can be made to represent all affine transformations and with the restriction that the matrices do not scale or shear along any axis, they will represent rigid transformations. Multiplication between two matrices represents the chaining of transformations and the transformation of points and vectors is computed with matrix-vector multiplication . While matrices are simple and efficient to use in a number of applications, they do not interpolate desirably without significant effort[2][25].

The core issue with matrix interpolation comes from the basic problem of rotation interpolation. From the results by Shoemake[24], it has been shown that unit quaternions can be used to get a rotation interpolation that is shortest path and constant speed. By combining a rotation representation with a vector to represent the translation, a quaternion-vector pair is formed. However a problem with this representation exists, as the basic interpolation methods inherited from quaternions are coordinate dependant when applied to these pairs. This is because the interpolation always assumes the centre of rotation to the be the origin of the current coordinate frame. Thus blending between transformations that have different centres in a new coordinate frame will cause problems. This is most notable in the area of mesh skinning, where bones will

[1] Any mention of rigid transformations will be referring specifically to proper rigid transformations of $\mathbb{R}^3$ since this is the vector space quaternions and dual quaternions act upon

almost always have different centres. A solution to this problem has been found, Spherical Blend Skinning[16], but it requires performing linear regression analysis to find the ideal centre, thus it is impractical in a real time application.

Dual Quaternions[8] present another representation that provides a solution to the issue of rotation centres[27][7][10]. They are constructed by considering the translation component embodied by a quaternion dual number. Multiplication following from both dual multiplication and quaternion multiplication lends directly to representing the chaining of transformations, and with the right representation for 3D points, one can transform points similarly to quaternions. Dual extensions of the quaternion interpolation methods are also naturally constructed, that have the same desired properties. Dual quaternions have been around since 1891, however there has been very little research on their use in computer science, save for some notable work by Kavan et al.[17][18][19] and various industry uses [21] [13].

The motivation for this project was then to understand what situations dual quaternions are most effective in. Previous work has looked at specific applications of dual quaternions as solutions to existing problems, but has not analysed their general usefulness. We wanted to know in what applications dual quaternions would be preferred, particularly in regards to real time transformation and interpolation.

From this we had three primary areas in the study. First we looked at the other rigid transformation methods and tested them. Secondly we investigated the rigid transformations mathematically to understand why the rigid transformation methods exhibit the properties we found from testing. Finally, we saw what the potential application of dual quaternions were after gaining an understanding of how they compare to other representations.

To gain an understanding of the properties that we wanted to compare, a basic skeletal animation system was built. The reason is that the skeletal system has three main areas that showcase many different important properties and methods. The bone structure was used to compare how hierarchies and complex chains of transformations can be combined together. Animation between transformations in these hierarchies shows the effectiveness of transformation interpolation. Finally vertex skinning is an important application of rigid transformation blending and implementing it showed the relative effectiveness of blending methods.

In the remainder of this report we analyse the different transformation representations, and gain an understanding of where dual quaternions fit into applications requiring rigid transformations. In Chapter 2 we take a more indepth look at the background behind rigid transformations, as well as a specific look at how they are used in skeletal animation. In Chapter 3 we outline the skeletal system that we built for this project, and present three parts of that system for which experiments were conducted to test the rigid transformation representations. In Chapter 4 we analyse the results and compare the rigid transformations, with a discussion about why certain representations may perform better than others. Finally Chapter 5 concludes the work while looking at future research avenues.

2 Related Work

In this chapter we look at the previous work relating to rigid transformations and their applications. In Section 2.1 we present a brief outline of quaternions. Section 2.2 then gives a description and an overview of the three interpolation methods, as well as some of the current findings with those methods. Finally in Section 2.3 we give a description of techniques required for skeletal animation, with particular emphasis on geometric skinning methods.

2.1 Quaternions

Quaternions, discovered by William Hamilton in 1843 [15], are a number system that extends the complex numbers by adding two additional elements. A quaternion q can be represented by the 4 real numbers w, x, y and z in the following form:

$$q = w + xi + yj + zk,$$

where $i^2 = j^2 = k^2 = ijk = -1$. From this it follows that $ij = k$ and $ji = -k$ so the complex elements are not commutative. Quaternion addition is done component-wise and multiplication is given by the Hamilton product. Given quaternions q_1 and q_2, the Hamilton product is defined as

$$q_1 * q_2 = \begin{array}{ll} (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2) & + \\ (w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2)i & + \\ (w_1y_2 - x_1z_2 + y_1w_2 + z_1x_2)j & + \\ (w_1z_2 + x_1y_2 - y_1x_2 + z_1w_2)k & \end{array},$$

which can be seen to be distributive multiplication over the elements. Due to the noncommutative nature of i, j, k, the Hamilton product is also noncommutative. Another important operation on quaternions is the conjugate q^*. This is an analog of the complex conjugate, and is given by

$$q^* = w - xi - yj - zk.$$

Of particular interest is unit quaternions, since they can represent the set of all rotational transformations in 3D space. A unit quaternion is a quaternion with a norm, or length, of 1, with the norm of q given by $||q||$ as

$$||q|| = \sqrt{w^2 + x^2 + y^2 + z^2}.$$

Given a unit quaternion q representing a rotation, and some point in 3D space $p = (x, y, z)$, the rotation of the point p by q is given by

$$w' + x'i + y'j + z'k = q(1 + xi + yj + zk)q^*,$$

where the transformed point is given by $p' = (x', y', z')$. It is important to note that both q and $-q$ give the same result, and thus represent the same rotation. Thus the set of unit quaternions does not uniquely contain all rotations and in fact for each rotation there are two quaternions that represent it. These rotations can also be chained together simply by multiplying them together, with $q_1 * q_2$ representing a rotation by q_2 followed by a rotation by q_1.

2.1.1 Rotation Interpolation

Unlike scale and translation interpolation which can be done as a simple linear combination, there is no single obvious method of rotation interpolation. As given by Bloom et al. in [3] when interpolating between two rotations, there are three key properties that are desired: shortest path, constant angular velocity and commutativity. Shortest path means that the interpolation takes the shortest path between the two rotations. Constant angular velocity means that for any constant change in the parameter of interpolation produces the same change in interpolated rotation. Commutativity means that when interpolating between multiple rotations in a sequence, the order that the interpolation is done in does not affect the result.

As shown in [3], due to the topological nature of the rotation space, no method of interpolation can satisfy all three of these properties. However, unlike matrix or Euler angle methods, the quaternion representation does provide a means to interpolate between two rotations that guarantees a shortest path rotation (provided that the correct choice of q or -q is chosen). Two methods of quaternion rotation interpolation exist that determine what other property they have, those being Spherical Linear Interpolation (Slerp) and Normalized Linear Interpolation (Nlerp).

Slerp, first introduced in [24] by Shoemake, is a method of interpolation over the 4th dimensional hypersphere of length 1 (the unit hypersphere). The unit quaternions are considered as points on the unit hypersphere and the interpolation path is the shortest path between the points over that hypersphere. A simple 2 dimensional analogy can be seen in Figure 2.1. Slerp does provide constant angular velocity since the angle between quaternions is what is linearly interpolated. It does not however provide commutativity as the relative angle change between the interpolated quaternions will change depending on the order, and thus the Slerp result will be different.

Slerp can be calculated for two quaternions q_1 and q_2 as

$$\text{Slerp}(q_1, q_2; t) = \frac{\sin(1-t)\theta}{sin\theta} q_1 + \frac{\sin t\theta}{sin\theta} q_2,$$

where $\cos\theta = q_1 \cdot q_2$ and with some interpolation parameter $t \in [0, 1]$. Because of the fact that q and $-q$ represent the same rotation, this can lead to the interpolation producing a path that has a greater than 180° rotation, that is, going the long way round. Negating one of the quaternions

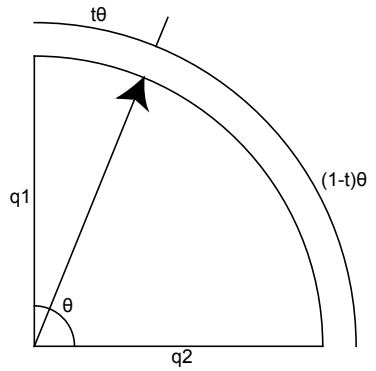


Figure 2.1: Slerp

solves this problem, and checking for which pair the dot product of the quaternions (considered as vectors) is positive finds whether one of the quaternions needs negating to use. This is because any spherical path with angle greater than 90° on the unit hypersphere will correspond to a rotation the long way around, and negative dot products will correspond to angles outside $[-90^\circ, 90^\circ]$.

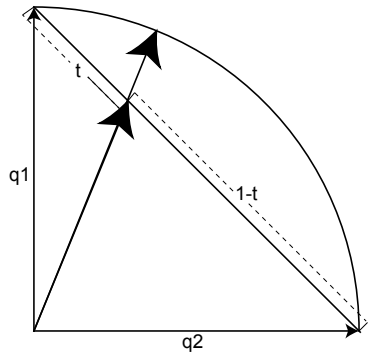


Figure 2.2: Nlerp

Nlerp[9][4] is defined for two quaternions q_1 and q_2 as

$$\text{Nlerp}(q_1, q_2; t) = \frac{q_1(1-t) + q_2 t}{||q_1(1-t) + q_2 t||},$$

given some interpolation parameter $t = [0, 1]$. It is a linear combination of the two quaternions, that is normalized to ensure it is unit length. This linearity provides commutativity but the projection back onto the hypersphere that is caused by the normalization will not have constant speed. As can be seen in Figure 2.2, the result is an identical path to Slerp, but the position along that path at a different parameter is different, thus resulting in a non-constant angular velocity. The same potential negating that is applied to one of the quaternions in Slerp is again done in Nlerp to ensure that the rotation is performed along the shorter path.

2.2 Rigid Transformations

A rigid transformation[22] is a transformation of a vector space that preserves the distance between any two points before and after modification [1] of the point by the vector. Proper rigid transformations also preserve the handedness, disallowing reflections, and these can be considered as the combination of a rotational transformation and a translation.

While many representations of rigid transformations exist, the focus of this research is on the application of dual quaternions, thus we will be examining those in comparison to common existing representations. Those representations are rigid transformations matrices, and quaternion-vector pairs.

2.2.1 Quaternion-Vector Pair

As quaternions are an effective method of representing rotations and translations can be represented by a vector, a combination of a quaternion and a translation vector is a natural representation of a rigid transformation. The transformation of a point p to the point p' with a quaternion-vector pair (q,v) is

$$p' = qpq^* + v$$

with the assumption that p is converted to a quaternion form so qpq^* produces the rotated vector. This applies the rotation to the point, and then adds the translation vector after. Given two quaternion-vector pairs, (q_1,v_1) and (q_2,v_2), applying (q_2,v_2) and then (q_1,v_1) to a point p gives

$$\begin{aligned} p' &= q_1(q_2pq_2{}^* + v_2)q_1{}^* + v_1 \\ &= q_1q_2pq_2{}^*q_1{}^* + q_1v_2q_1{}^* + v_1 \end{aligned}$$

This means that the chaining of two quaternion-vector pairs (q_1,v_1) and (q_2,v_2) is given by

$$(q_1,v_1) * (q_2,v_2) = (q_1 * q_2, q_1v_2q_1{}^* + v_1)$$

The simplest method of interpolation is to linearly interpolate the translation component, and to interpolate the quaternion component using one of the rotational interpolation methods from the previous section. Thus you have Slerp and Nlerp defined (for quaternion-vector pairs (q_1,v_1) and (q_2,v_2)) as

$$\begin{aligned} \text{Slerp}((q_1,v_1),(q_2,v_2);t) &= (\text{Slerp}(q_1,q_2,t),\text{Lerp}(v_1,v_2,t)) \\ \text{Nlerp}((q_1,v_1),(q_2,v_2);t) &= (\text{Nlerp}(q_1,q_2,t),\text{Lerp}(v_1,v_2,t)). \end{aligned}$$

However, problems may be encountered because this interpolation is coordinate dependant, and more specifically it is not right-transitive[16][17]. What this means is, changing the basis for

[1] Modification / modify will be used to refer to the act of transforming a point by a transformation, to avoid confusion with the word transformation

coordinates before and after interpolation, changes the results of the interpolation, due to a difference in the centre of rotation.

While these problems with changes in basis do prevent it from being useful in all applications, it is likely very commonly used. This is because applications such as 3D editors and video games will store key frame data for rotation and translation independently, so it can be interpolated and changed independently, and when the rotation data is stored as a quaternion, the result is that rigid transformations, in effect, are stored as a quaternion-vector pair.

2.2.2 Rigid Transformation Matrices

A transformation matrix is a matrix that transforms a vector by translation, scaling, shearing, rotation and/or reflection. A transformation matrix for $\mathbb{R}^3$ is constructed as a 4x4 matrix and the transformed point is represented in homogeneous coordinates. The transformation from x to x' is then

$$\begin{bmatrix} x' \\ 1 \end{bmatrix} = \begin{pmatrix} A & v \\ 0,\ldots,0 & 1 \end{pmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix},$$

where A is a 3x3 linear transformation matrix and v is a translation vector. A rigid transformation matrix is a transformation matrix where the linear transformation is restricted to an orthogonal transformation, thus only allowing translation and rotation. A rigid transformation is represented by a matrix R is then

$$R = \begin{pmatrix} Q & v \\ 0,\ldots,0 & 1 \end{pmatrix},$$

where $Q^T Q = I$. Due to the associativity of matrix multiplication, a sequence of rigid transformation matrices can be chained together through matrix multiplication. Thus $R_1 * R_2$ represents a transformation by R_2 followed by a transformation by R_1.

The simplest method of interpolation between two rigid transformations using matrices is a component-wise linear interpolation of the matrices. Thus for some parameter t and matrices R_1 and R_2,

$$\text{Lerp}(R_1, R_2; t) = R_1(1-t) + R_2 t.$$

There are some notable issues with this method. Basic matrix linear interpolation introduces scaling artifacts into the rotations, making the 3x3 matrix no longer orthogonal, and the rotation interpolation has neither constant angular velocity nor is it a shortest path interpolation.

Orthonormalization of the rotation matrix can remove the first issue, but this requires a slow iterative process to get good results and does not provide the other desired rotation properties. Another method for interpolating rotation matrices is log-matrix blending[2]. In this case instead the logarithms of the matrices are linearly interpolated, and the result is exponent of this interpolation. This fixes the problem with constant angular velocity but does not guarantee shortest path and due to the calculation of matrix logarithms and exponentials, it is significantly slower.

Currently the main advantage of representing rigid transformations as matrices is the increased speed of transforming points, since each value in the matrix only needs to be multiplied

by one vector value. They are also are the most supported rigid transformation representation in graphics APIs, so they are generally simpler to use. However as shown by many others, the interpolation issues reduce its effectiveness in areas such as skinning where more accurate blending is a must.

2.2.3 Dual Quaternions

Dual numbers[8] are a number system that extends the real numbers by adding the element ε with $\varepsilon^2 = 0, \varepsilon \neq 0$. Thus we have addition and multiplication of dual numbers defined by

$$(a_1 + b_1\varepsilon) + (a_2 + b_2\varepsilon) = (a_1 + b_1) + (a_2 + b_2)\varepsilon$$
$$(a_1 + b_1\varepsilon)(a_2 + b_2\varepsilon) = a_1b_1 + (a_1b_2 + b_1a_2)\varepsilon.$$

Dual quaternions are then an extension of quaternions with the same ε element. Addition and multiplication, similar to dual numbers, is given by

$$(q_{r1} + q_{d1}\varepsilon) + (q_{r2} + q_{d2}\varepsilon) = (q_{r1} + q_{r2}) + (q_{d1} + q_{d2})\varepsilon$$
$$(q_{r1} + q_{d1}\varepsilon)(q_{r2} + q_{d2}\varepsilon) = q_{r1}q_{r2} + (q_{r1}q_{d2} + q_{d1}q_{r2})\varepsilon.$$

Analogous to quaternions, there is the concept of the quaternion conjugate $d^* = q_r^* + q_d^*\varepsilon$, and in addition there is the dual number conjugate, given by $\overline{d} = q_r - q_d\varepsilon$. Also, like quaternions, there is the norm of a dual quaternions $||d||$ which is calcuated as

$$||d|| = \sqrt{(}dd^*)$$

Unit dual quaternions are the set of all dual quaternions with a norm of one. Because the norm of a dual quaternion is a dual number, $q_r^*q_d + q_d^*q_r = 0$ must be true for all unit dual quaternions.

The set of all unit dual quaternions can be used to represent rigid transformations[27][10]. A rotation by a unit quaternion q, plus a translation by a vector represented as quaternion v gives a dual quaternion $d = q + \varepsilon\frac{v.q}{2}$. Then, given a unit dual quaternion d and a point $p = 1 + (xi + yj + zk)\varepsilon$, the point is transformed by this transformation by

$$1 + (x'i + y'j + z'k)\varepsilon = dp\overline{d}^*, \qquad \overline{d}^* = q_r^* - q_d^*.$$

Dual quaternion transformations can be chained together with multiplication, and inverse transformations are represented by the quaternion conjugate d^*. As with other rigid transformation representations, $d_1 * d_2$ represents a transformation by d_2 followed by a transformation by d_1.

Interpolation of dual quaternions can be extended from quaternion methods of interpolation. Thus we have an Dual quaternion Linear Blending (DLB), an extension of Nlerp, and Screw Linear Interpolation (Sclerp), an extension of Slerp[17][18]. Screw Linear interpolation involves interpolating around a screw motion. A screw involves a rotation around an axis, and a translation along that same axis. Sclerp then is a linear interpolation on the amount of translation on the

screw axis, and the angle of the screw axis. This can be calculated, similarly to quaternion Slerp, given some interpolation parameter t and two dual quaternions as

$$\text{Sclerp}(d_1, d_2; t) = d_1 * (\cos t\frac{\alpha}{2} + \sin t\frac{\alpha}{2}u).$$

where $t\alpha$ is a dual number representing both the angle around the screw axis and translation on the screw axis u. Like Slerp, this method is shortest path and has constant angular velocity. DLB extends Nlerp to dual quaternions straightforwardly through

$$\text{DLB}(d_1, d_2; t) = \frac{d_1(1-t) + d_2 t}{||d_1(1-t) + d_2 t||},$$

This method use the same screw axis as Sclerp so it is also has shortest path rotational interpolation, but it interpolates differently and trades constant angular velocity for commutativity. It is also simpler and more efficient to implement, as it does not require the calculation of the α angle or dual sine and cosine implementations.

Both of these interpolation methods are left and right distributive, so they are coordinate invariant and thus do not have the same issues with changes in centre of rotation. It is important to note, that like quaternions, there are two dual quaternions representing each rotation. That is, d and $-d$ represent the same rotation, so in order to get a shortest path, a check must be made to find which dual quaternion to interpolate with.

2.3 Skeletal Animation

Animation of deformable figures has many applications, most importantly in digital film and video game production. These figures could be humanoid, animal, or even just a deformable object such as cloth. While several methods exist for providing animation to these figures, one method that has seen prominent use is skeletal animation [5]. Skeletal animation, in its simplest form, consists of a skeleton, made up of a hierarchy of transformations known as bones. These bones each effect some part of the figure. Animation consists of interpolating between different transformations of the bones, called poses.

While scaling can be applied to areas affected by bones, it is common to use rigid transformations as this constraint is natural and makes the bone hierarchy simpler. When bones are rigid transformations, rigid transformations then become a large part of skeletal animation, thus the representation used becomes critically important to the function of the skeleton. A change in representation changes how the world transformation for bones is calculated, how skeletal poses are blended together to create animation, and how the figure is deformed based on these transformations.[12]

For the course of this paper, we will be only be considering skeletal animation as it relates to polygonal meshes consisting of vertices, edges and faces.

2.3.1 Geometric Skinning

When considering a polygonal mesh, each vertex is assigned to one or more bones and given a weight on those bones. Skinning is the calculation of the transformed vertex positions based on the assigned bones, and geometric skinning refers to the process of blending the transformations through some method, and transforming the vertex to this blended transformation. The general formula for this is, given transformations $r_1 \dots r_n$ and convex weights $w_1 \dots w_n$,

$$v' = blend(r_1, w_1, \dots, r_n, w_n) * v.$$

When using rigid transformation matrices to represent the transformations, using a linear combination of the matrices as the blending method is known as Linear Blend Skinning. This is an extension of linear interpolation of matrices to support more than two transformations. The name linear skin blending is due to the fact that the transformation method is linear, and thus this blending can be done before or after the transformation of the original points. That is for transformation matrices $M_1 \dots M_N$ the skinning is given by

$$v' = (\sum_{i=1}^{n} w_i M_i) * v = \sum_{i=1}^{n} (w_i M_i * v).$$

A major issue with this type of blending, is like linear matrix interpolation, the resulting blended transformation will not actually be a rigid transformation, and the rotational blending will not be shortest path. This first factor causes a problem known as the candy wrapper effect, where the mesh folds into a single point due to the scaling.

One of the solutions to this rigid transformation problem is log-matrix blending. The exponential of a matrix M is given by

$$e^M = \sum_{i=0}^{\infty} \frac{M^i}{i!}$$

and from this the definition of the logarithm $\log M$ is just the inverse function of this. While not all matrices have a logarithm, all invertible matrices, and thus all rigid transformation matrices will have a logarithm. The formula then, for log-matrix blending transformation matrices $M_1 \dots M_N$ with convex weights $w_1 \dots w_n$ is

$$v' = e^{\sum_{i=1}^{n} w_i \log M_i} * v.$$

This preserves the scaling, so will always result in a rigid transformation. However it is not shortest path, and can lead to artifacts that are much more apparent than those with linear blend skinning. In addition the computational cost of calculating matrix logarithms and exponentials is high, so in real time situations, it is often infeasible to use the matrix logarithms technique.

It is possible just to linearly blend quaternion-vector pairs using an extension of Nlerp for more than two transformations. This method is known as direct quaternion blending DLB, however, the coordinate dependance of this linear blending produces artifacts when skinning. This is because the rotation centre for this linear blending is always the origin which in the case of

a model will likely be its centre of mass, when in fact the centre of rotation should be on or near one of the joints. Spherical blend skinning[16] is a proposed method to properly blend quaternion-vector pairs for skinning by taking into account the issues with rotation centres. It does this by calculating an appropriate centre of rotation for each set of transformations. This centre is defined as the point that has the least variation after being transformed by each of the blending transformations. This can be found using a least squares solution to the system of equations of each of the transformations. The vertex is then transformed such the transformation occurs around this centre. The method fixes the blending artifacts caused by straight linear blending of quaternion-vector pairs, but because of the least squares calculation, it has a significant computational cost.

Blending of dual quaternions presents another method of geometric skinning[17]. Blending with a method akin to Sclerp but generalized for any number of transformations is possible, but it is much more complex than other blending methods and because to the noncommutativity of dual quaternions, it would depend on the order that the elements were blended. Using DLB is a much better idea, as this generalizes simply as

$$\mathrm{DLB}(d_1, w_1, \ldots, d_n, w_n) = \frac{d_1 w_1 + \cdots + d_n w_n}{||d_1 w_1 + \cdots + d_n w_n||},$$

for dual quaternions $d_1 \ldots d_n$. Thus one can transform a vertex v by dual quaternions transformations with convex weights $w_1 \ldots w_n$ simply as

$$v' = \frac{d_1 w_1 + \cdots + d_n w_n}{||d_1 w_1 + \cdots + d_n w_n||} v,$$

This has all the properties of DLB blending, so will produce a valid transformation, interpolate along the shortest path and will be coordinate invariant. Because it is coordinate invariant, it must then handle rotation centres correctly and so dual quaternion blending avoids the artifacts produced by bad rotation centres.

2.3.2 Spline Interpolation

A spline is a piecewise defined polynomial function, whose derivatives are continuous up to a certain order[1]. An interpolation spline is a spline function that passes through an ordered list of points, in that same order. In the context of animation, spline interpolation is then given a list of points and a parameter t representing time, calculate the spline function f and compute $f(t)$. In this case there may be a spacing parameter between points, that determines the amount of time between two points on the curve. Given a sufficient order of smoothness, this can be used to animate a smooth motion through a series of points.

One type of spline with many applications is the cubic hermite spline[6]. In this case a component, on an interval $(0,1)$ of the spline, is specified in terms of four components and given by

$$f(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 + 3t^2)p_1 + (t^3 - t^2)m_1$$

In this function, p_0 is the start point, p_1 is the end point, m_0 is the tangent vector at the start point and m_1 is the tangent vector at the end point. For the same spline over the interval (t_k, t_{k+1}), transforming to $f(\frac{t-t_k}{t_{k+1}-t_k})$ will give that interval, however to achieve the same motion, the tangents will have to be scaled to the size of the interval. A combination of these components at connected intervals will form the full spline from which a point can be found by a certain parameter t.

A simple type of hermite spline used extensively in computer graphics is the Catmull-Rom spline[26]. In this case the tangent vector at each point is chosen to be the vector from the previous point to the next point.

$$m_k = \frac{p_{k+1} - p_{k-1}}{t_{k+1} - t_{k-1}}$$

This method can be used to simply construct a smooth keyframed motion from a list of vector values and the time, or t value, when they are at that position.

3 Implementation and Results

In this chapter we present the methods by which the components of the skeletal animation system were built and the how the properties of the transformations were examined. The end result of the skeletal system was to show a potential application of dual quaternions, as well as having something through which the properties of various rigid transformations were able to be compared in a meaningful way. The bone hierarchy contains the transformation properties of rigid transformations and the way the transformations chain, keyframed animation shows interpolation between the transformations and hierarchies and vertex skinning shows the blending properties. The focus is on real time applications, so certain methods that are known to be inefficient, such as spherical blend skinning, were not compared.

Section 3.1 Discusses the implementations of the rigid transformation. In Section 3.2 we discuss how the skeletal animation system that is used for testing was built. Section 3.3 analyses the memory and space efficiency of each transformation representation. Finally in the other sections we show our examination of some applications of the three transformations: matrices, quaternion-vector pairs and dual quaternions. These applications are in order: memory requirements, transformation hierarchies, interpolation of hierarchies and geometric skinning. In each of these other sections we show how this was implemented or tested, give a mathematical analysis of the specific application, and provide results on how each type of transformation, or method on the transformation performs.

3.1 Rigid Transformation Implementation

Almost all of the mathematical operations were implemented in terms of the Open GL mathematics library, glm. The matrices are implemented in terms of glm::mat4 (4x4 matrices) with added functions for blending and interpolation. The quaternion-vector pair is implemented as a glm::quat (quaternion) and a glm::vec3 (3D vector) for the translation. Functions for combining chains of these pair, as well as the modification functions and the interpolation functions were created. The dual quaternions were constructed in terms of two quaternions, with the multiplication, addition and conjugate functions implemented, based on the methods in [23] and [20]. Sclerp and DLB interpolation methods were implemented, based on the functions by Kenwright in [21].

3.2 Skeletal Animation System

The skeletal animation system we constructed is composed of three parts, each using rigid transformations. The first is the bone structure, which is a hierarchy of the transformations that define each bone and are themselves dependent on the parent bones transformation. The second part is the skeleton animations, which is generated from interpolation between different hierarchies, each of these hierarchies being a keyframe of the animation. The final part is the skinning, where vertices are mapped to bones, and then transformed based on the bones they are mapped to. The structure of the full system can be seen in Figure 3.1. We constructed three different kinds of skeletal system, based on the three rigid transformation representations being analysed, each using the same structure.

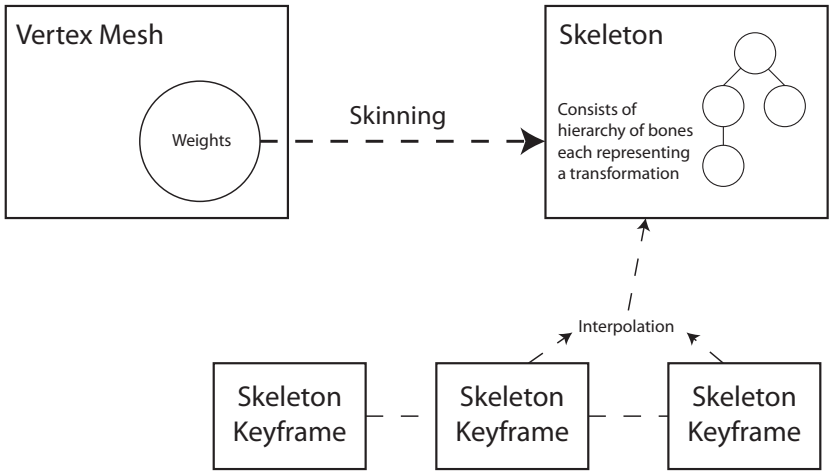


Figure 3.1: Skeletal Animation System Structure

3.2.1 Bone Hierarchy

The skeleton object consists of a number of bones, assembled in a hierarchy. The root of the hierarchy of bones has the rigid transformation of the object that the skeleton is attached to. The rest of the bones each have a parent, any number of children, and a transformation relative to the parent. A basic skeleton structure can be seen in Figure 3.2. As an example, the transformation for Bone 3 relative to the root node would be $r_1 * r_2 * r_3$, where $*$ is the method used to chain two transformations.

Because transforming a point to a bone requires transforming not just by the bone but also by all the parents of the bone, the operation can be computationally expensive. Thus to reduce the workload, the skeleton structure also contains a list of the transformations of each bone relative to the root node, that is updated whenever there is a change in any of the bones transformations. This is calculated in Algorithm 1 by traversing the tree starting at the root and chaining transformations. Due to the associativity of modification and the chaining operation across all rigid

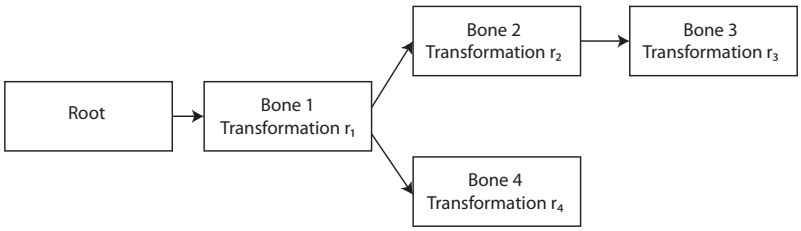


Figure 3.2: Example Bone Hierarchy

transformations, modification of a point by this final transformation is equivalent to a sequence of modifications on the point.

Algorithm 1: Calculate Node Transformation Relative To Root

Input: The bone B
Set the root transform of B to the root transform of the parent of B, times the relative transform of B;
for *each child bone of B* **do**
  Recursively run this algorithm on the child bone
end

3.2.2 Skeleton Animation

There are two types of keyframed animation in the skeleton system. The first is simple straight interpolation between two transformation hierarchies. For this we take two skeletons with the same hierarchical structure as input, a value t as the interpolation parameter (the time between the two keyframes) and from this we get a third skeleton. The method, shown in Algorithm 2 is used to interpolate the various transformations. The interpolation method used in our application is linear interpolation for matrices, and nlerp for dual quaternions and quaternion-vector pairs.

Algorithm 2: Interpolation of Skeleton Keyframes

Input: Two skeletons S_1 and S_2 with identical structure and the interpolation parameter t
Output: The interpolated skeleton S_3
Create the structure of S_3 with identical structure to S_1 and S_2 ;
for *Each bone* B_1 *of* S_1 **do**
  Take the corresponding bone B_2 of S_2;
  Take the transformation r_1 of B_1 and r_2 of B_2;
  $r_3 = Interpolate(r_1, r_2, t)$;
  Set the transformation of the bone B_3 to r_3;
end

It is important to note that the interpolation is performed on the relative transformation of each bone. While it is possible to chain the transformations using the above Algorithm 1 and then interpolate these values with Algorithm 2, this does not produce the correct interpolation as the interpolation will not occur around the correct centre. As can be seen in Figure 3.3 the result when doing this causes a rotation around a different center, producing a different interpolation.

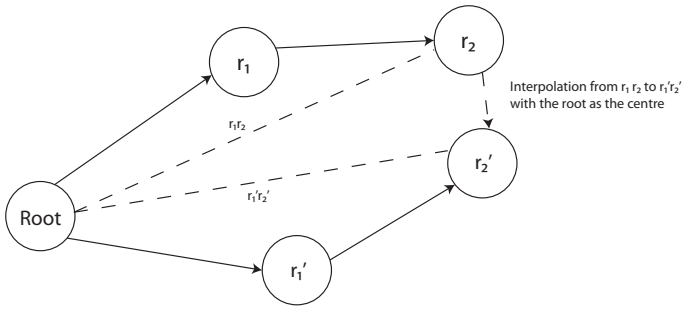


Figure 3.3: Interpolation after combining transformations

The other kind of keyframed animation uses spline interpolation. In this we treat the transformations of bones as vectors in n-dimensional space, and then generate a spline between them, and in this case the spline we are using in the applications is a catmull-rom spline. Since the vector is constructed from the elements within the transformations, matrices are 12 dimensional, dual quaternions are 8 dimensional and quaternion-vector pairs are 7 dimensional. The algorithm for spline animation is given by Algorithm 3

Algorithm 3: Spline Interpolation of Skeleton Keyframes

Input: List of structurally identical skeletons $S_1 \dots S_n$, the time values $t_1 \dots t_n$ and the interpolation parameter p

Output: The interpolated skeleton S_t

Find k such that $t_k \leq p < t_{k+1}$;

$t = p - t_k$;

Create the structure of S_t with identical structure to S_k;

for *Each bone* B_k *of* S_k **do**

Take the corresponding bones of S_{k-1}, S_{k+1}, S_{k+2};

Take the transformation r_j of each B_j found above;

$m_k = (r_{k+1} - r_{k-1})/(t_{k+1} - t_{k-1})$;

$m_{k+1} = (r_{k+2} - r_k)/(t_{k+2} - t_k)$;

$r_t = (2t^3 03t^2 + 1)r_k + (t^3 - 2t^2 + t)m_k + (-2t^3 + 3t^2)r_{k+1} + (t^3 + t^2)m_{k+1}$;

Take the corresponding bone B_t of S_t;

Set the transformation of the bone B_t to r_3;

end

An issue with this method is that the resulting transformations are not rigid transformations. This can be fixed by computing the weight value on each of the four transformations r_{k-1}, r_k, r_{k+1}

and r_{k+2} as above, and using some blending method on these transformations. In our applications we use DLB for dual quaternions and Nlerp for the quaternion part of quaternion-vector pairs. Because the spline interpolation is an extension of blending algorithms used in blending, no specific experiments for it were conducted.

3.2.3 Vertex Skinning

The skeleton is connected to a vertex mesh through bone-vertex weights. These weights are the basis for the vertex skinning system. For any transformation representation and using some blending method blend for that transformation, the skinned vertex is as in Section 2.3.1,

$$v' = blend(r_1, w_1, \ldots, r_n, w_n) * v,$$

with the weight w_k as the weights that vertex v has for bone b_k with transformation r_k. This transformation is the transformation that is relative to the root, precalculated in Algorithm 1. In our implementation the blending method used is linear blending for matrices, DLB for dual quaternions and Nlerp for quaternion-vector pairs.

3.3 Memory Analysis

A simple counting can be made between the transformations. Ignoring the bottom row of the matrix, as it is always $(0,0,0,1)$ we can see the number of floating points for each of the transformation types in Table 3.1. Further optimisations can be made at the extent of comparisons, as is seen in [13] by only storing the quaternions with 3 of the elements. This is due to the constraint that the sum of the square of the elements must add to 1, and also due to the property that q and $-q$. Thus the fourth element can be computed simply as $w = \sqrt{1-(x^2+y^2+z^2)}$

Rigid Transformation Matrices	Quaternion-Vector Pair	Unit Dual Quaternions
12	7	8

Table 3.1: Comparison of Memory for Transformation Representations

3.4 Transformation Hierarchies

3.4.1 Method

The goal of this experiment was to compare the effectiveness of the representations at handling chains of transformations, both in terms of accuracy and efficiency. We looked at the speed of combining transformations and modifying points by these transformations, as well as the accuracy of the applying the transformation to a point. We investigated not only how the different

representations hold up just as straightforward representations, but also how they work specifically in the context of hierarchies.

Two tests were setup to measure the efficiency of the representations. The first checks the speed of chaining transformations. In it we have a chain of six transformations, and calculate the time taken to combine all the transformations in the chain together for each representation. The second is comparing the speed it takes to modify a point by the transformation. In this experiment we simply calculated the time for the $r * p$.

There were also three tests concerning the accuracy of the representations. The first was just simply examining the difference in the translation component, by modifying a zero vector $(0,0,0)$ by the rigid transformation. This gave us information on whether the representation distorts the translation. We also had a test to see what the accuracy drift is in a transformation chain. For this we calculated the value of the transformation of a point successively by each transformation in the chain. A final test showed the result of combining the transformation chain into a single transformation and then modifying the point by this transformation. This was compared with the successive transformation results to see the difference in the methods.

3.4.2 Mathematical Analysis

For the simplest optimisations we can ignore doing any multiplication by one, any multiplication by zero, and any addition by zero. With these optimisations we calculate the number of addition and multiplication operations required to modify vectors and points given by Table 3.2, as well as the number of operations to chain the transformations together, given in Table 3.3.

	Adds	Multiplys	Total Operations
Rigid Transformation Matrices	9	9	18
Quaternion-Vector Pair	18	15	33
Unit Dual Quaternions	47	45	92

Table 3.2: Comparison of Operations for Modifying Points with Transformations

	Adds	Multiplys	Total Operations
Rigid Transformation Matrices	27	36	63
Quaternion-Vector Pair	30	31	62
Unit Dual Quaternions	48	36	84

Table 3.3: Comparison of Operations for Chaining Transformations

We can also see that forming a chain of transformations is associative and modifying a point with this transformation is associative. We can see that it simply follows for matrices M_1, M_2

and M_3 with a point $p = (x, y, z, 1)$ that

$$M_1 * (M_2 * (M_3 * V)) = (M_1 * M_2) * (M_3 * V) = (M_1 * M_2 * M_3) * V$$

The associativity of quaternions and dual numbers follows from quaternions. It can also easily be shown that the transformation of point $p = 1 + xi + yj + zk$ by dual quaternions d_1 and d_2 is associative by

$$(d_1 d_2)p(d_1 d_2)^* = d_1 d_2 p d_2^* d_1^* = d_1(d_2 p d_2^*)d_1^*.$$

This follows from the quaternion property $(q_1 q_2)^* = q_2^* q_1^*$. The quaternion-vector air associativity can be seen for chaining by

$$\begin{aligned}
(q_1, v_1) * ((q_2, v_2) * (q_3, v_3)) &= (q_1, v_1) * (q_2 q_3, v_2 + q_2 v_3 q_2^*) \\
&= (q_1 q_2 q_3, v_1 + q_1(v_2 + q_2 v_3 q_2^*)q_1^*) \\
&= ((q_1 q_2)q_3, (v_1 + q_1 v_2 q_2^*) + (q_1 q_2)v_3(q_2^* q_1^*)) \\
&= (q_1 q_2, (v_1 + q_1 v_2 q_2^*)) * (q_3, v_3) \\
&= ((q_1, v_1) * (q_2, v_2)) * (q_3, v_3)
\end{aligned}$$

where (q_n, v_n) are quaternion-vector pairs. It also holds for modification of some point $p = (x, y, z)$ by (q_1, v_1) and (q_2, v_2) since

$$\begin{aligned}
(q_1, v_1) * ((q_2, v_2) * p) &= (q_1, v_1) * (v_2 + q_2 p q_2^*) \\
&= v_1 + q_1(v_2 + q_2 p q_2^*)q_1^* \\
&= v_1 + q_1 v_2 q_1^* + q_1 q_2 p q_2^* q_1^* \\
&= ((q_1, v_1) * (q_2, v_2)) * p
\end{aligned}$$

These properties confirm that all three representations are in fact representations of rigid transformations.

3.4.3 Results

Table 3.4 shows the timing results for chaining of transformations in seconds. Each of these results show the timing for six transformations combined 100000 times. As can be seen, the timing shows that the dual quaternion implementation is much faster at combining than the quaternion-vector pair and matrix implementation. However looking at the number of operations, these results should be similar. The matrix code just uses standard matrix math on 4x4 matrices and thus makes no optimisations for the fact that the last row is always $(0, 0, 0, 1)$. This could be why it is slower than it should be.

The timing results for modification of a point by a transformation are shown in Table 3.5, with each modification performed 1000000 times. Comparison with the number of operations calculated seems to suggest that the implementation of matrices is supposed to be the fastest. Even given, as above, that the implementation uses 4x4 matrices, it still should not make the

	Test 1	Test 2	Test 3	Test 4	Test 5	Mean
Rigid Transformation Matrices	1.65	1.64	1.64	1.64	1.64	1.64
Quaternion-Vector Pair	0.31	0.31	0.31	0.31	0.31	0.31
Unit Dual Quaternions	0.18	0.17	0.17	0.17	0.18	0.17

Table 3.4: Running Time for Combining Chained Transformations

	Test 1	Test 2	Test 3	Test 4	Test 5	Mean
Rigid Transformation Matrices	1.42	1.42	1.44	1.44	1.42	1.43
Quaternion-Vector Pair	0.47	0.47	0.48	0.47	0.47	0.47
Unit Dual Quaternions	1.11	1.11	1.12	1.12	1.11	1.11

Table 3.5: Running Time for Modifying Points with Transformations

operation 3 times slower than quaternion-vector pairs. Regardless, we can clearly see that dual quaternions are much slower than using a regular quaternions with a translation vector.

The first test on the accuracy was comparing the effect of pure translation. In this case three rigid transformations were tested and the results can be seen in Table 3.6. The transformation used for these results was given by a translation of $(1,0,0.5)$, and rotation around $(0,1,1)$ by 80 degrees. While there are no large inaccuracies, there is a minor divergence in the dual quaternion case. This can likely be attributed to the fact that the translational component is multiplied by the rotation, producing this small error.

	Result
Rigid Transformation Matrices	(1.00000000, 0.00000000, 0.50000000)
Quaternion-Vector Pair	(1.00000000, 0.00000000, 0.50000000)
Unit Dual Quaternions	(1.00000012, -0.00000003, 0.50000000)

Table 3.6: Results of applying translation component

The next test shows the accuracy of a series of transformations applied to a point. Each row in Table 3.7 shows a transformation, and the result of modifying the previous point with the transformation. This test is used to look at the accuracy of transforming something in a chain of transformations. After only four transformations, there is a distinct difference in the results. The matrix implementation and the dual quaternion implementation diverge by a factor of 10^-5, however the divergence increases with each transformation that is performed, which is relevant for chains of transformations.

We also have the results from the third accuracy test, that shows us the result of combining the chained transformations together and then modifying the point $(0,0,0)$ in Table 3.8. The

	Angle/Axis	Translation	Rigid Transformation Matrices
Trans A	$70°, (-0.089, 0.445, -0.89)$	$(2, 2, 2)$	2.0000000, 2.0000000, 2.0000000
Trans B	$70°, (-0.667, -0.333, -0.667)$	$(0, -2, -0.5)$	2.7737715, -0.5859429, 1.0187607
Trans C	$170°, (0.707, -0.707, 0)$	$(10, -1, -4)$	10.4774714, -3.8822441, -4.7346449
Trans D	$10°, (0.707, 0, 0.707)$	$(0, 0, 1)$	10.8386106, -1.9554013, -4.0957837

Quaternion-Vector Pair	Unit Dual Quaternions
2.0000000, 2.0000000, 2.0000000	2.0000005, 2.0000007, 2.0000002
2.7739086, -0.5862051, 1.0187545	2.7739096, -0.5862050, 1.0187547
10.4777184, -3.8823950, -4.7346096	10.4777203, -3.8823957, -4.7346106
10.8388376, -1.9558195, -4.0957294	10.8388424, -1.9558202, -4.0957308

Table 3.7: Results of performing chains of transformations

transformations used are the same as in the previous test, and in the same order, so a comparison can be made with the last row of Table 3.7. We see that there is some small difference in values, with matrices having less difference than the quaternions representations. However it is less than the differences between the representations, with the order of magnitude of difference being 10^-7 in the worst case.

Rigid Transformation Matrix	10.8386087, -1.9554017, -4.0957837
Quaternion-Vector Pair	10.8388386, -1.9558198, -4.0957308
Unit Dual Quaternions	10.8388443, -1.9558202, -4.0957317

Table 3.8: Results of combining transformations and then modifying a point

3.5 Interpolation of Hierarchies

3.5.1 Method

Interpolation is useful for animation and motion in 3D space, so it is important to consider the properties that various interpolation methods have. The goal for this experiment was to look at the interpolation between hierarchies and compare the effectiveness of the rigid transformation methods at performing interpolation. We investigated the efficiency of the methods, how they compared to each other, and what properties they possessed.

The first study we looked at is interpolation separate from the hierarchy. Specifically we gained an understanding of the properties of the representations when used with various interpolation methods. We analysed the different methods, did a test to look at the efficiency of the interpolation, and looked quantitatively at how the interpolation methods changed the interpolation. We considered the effects of the interpolation on the transformations, and what desirable properties the interpolation had.

We then looked at the effects interpolation has on rigid transformations in a hierarchy. We again performed some analysis on the interpolation of transformation chains mathematically, as well as looking at empirical results from our skeletal animation application. We looked at how being in a chain affected the interpolation methods, and what the cumulative effects were on the joint transformations.

3.5.2 Mathematical Analysis

Given two rigid transformations matrices M_1 and M_2, linearly interpolating these gives us $(1-t)M_1+tM_2$. However, this is not guaranteed to be a rigid transformation matrix. For instance consider

$$M = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

which represents a rotation of π around the axis $(0,0,1)$. Interpolating this with the identity, and with $t = 0.5$, gives us

$$(1-0.5)M+0.5I = 0.5(M+I) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

which is not a rigid transformation. Alexa [2] produced a solution to the problem with blending matrix logarithms. Thus with two matrices M_1 and M_2, the interpolation is given by

$$logmat(M_1,M_2,t) = \exp((1-t)\log(M_1)+t\log(M_2))$$

However the computational cost of computing matrix logarithms is high as it is an iterative process. This method still does not fix the other issue with matrices, which is that the interpolation does not follow the shortest path of rotation.

Quaternion-vector pairs can use the shortest path interpolation algorithms from quaternions, and interpolate linearly between the transformations to achieve better interpolation. This interpolation is coordinate dependant however, as it is not right distributive, which is shown below.

$$\begin{aligned}
&\text{Nlerp}((q_1,v_1),(q_2,v_2),t)*(q_3,v_3) \\
&= (\text{Nlerp}(q_1,q_2,t),(1-t)v_1+tv_2)*(q_3,v_3) \\
&= (\text{Nlerp}(q_1,q_2,t)q_3,(1-t)v_1+tv_2+\text{Nlerp}(q_1,q_2,t)v_3\,\text{Nlerp}(q_1,q_2,t)^*) \\
&= (\text{Nlerp}(q_1q_3,q_2q_3,t),(1-t)v_1+tv_2+\frac{((1-t)q_1+tq_2)v_3((1-t)q_1+tq_2)^*}{||(1-t)q_1+tq_2||})
\end{aligned}$$

However $\text{Nlerp}((q_1,v_1)*(q_3,v_3),(q_2,v_2)*(q_3,v_3),t)$ is

$$\begin{aligned}
&\text{Nlerp}((q_1,v_1)*(q_3,v_3),(q_2,v_2)*(q_3,v_3),t) \\
&= (\text{Nlerp}(q_1q_3,q_2q_3,t),(1-t)(v_1+q_1v_3q_1^*)+t(v_2+q_2v_3q_2^*)) \\
&= (\text{Nlerp}(q_1q_3,q_2q_3,t),(1-t)v_1+tv_2+(1-t)(q_1v_3q_1^*)+t(q_2v_3q_2^*))
\end{aligned}$$

However looking at the second part of the translation we see that,

$$(1-t)(q_1 v_3 q_1^*) + t(q_2 v_3 q_2^*) \neq \frac{((1-t)q_1 + tq_2)v_3((1-t)q_1 + tq_2)^*}{||(1-t)q_1 + tq_2||},$$

so the translation will be different. This can be understood to be a change in the centre of rotation. This may not be a problem when it doesnt matter or you need the centre to be the origin of the current coordinate frame. However when you want the centre to produce an ideal motion you need the centre to have minimal translation from the transformations. This centre can be found for the quaternion-vector pairs, but is generally slow. For dual quaternions, it naturally interpolates with this ideal centre and we get DLB as an extension of Nlerp and ScLerp as an extension of Slerp.

The work by Kavan et al. in rigid transformation blending[19] analysed the difference between Sclerp and DLB and found that the the upper bounds for rotations difference were 8.15 degrees and 15% translation. The difference will tend to be smaller than this, but in transformations chains, the difference could compound.

3.5.3 Results

We produced results looking at the amount of time 1000000 interpolations took in Table 3.9. However as in the previous section, the matrix results are likely skewed from an ideal implementations as the matrix uses all 16 values as opposed to only the 12 that are needed. We see that the interpolation using Sclerp is almost twice as slow as the DLB implementation, and that the respective quaternion-vector pair methods are slightly faster than the dual quaternion methods.

	Test 1	Test 2	Test 3	Test 4	Test 5	Mean
Rigid Transformation Matrices	3.02	2.97	2.97	2.98	2.97	2.98
Quaternion-Vector Pair Nlerp	0.97	0.97	0.97	0.99	0.97	0.97
Quaternion-Vector Pair Slerp	1.30	1.29	1.29	1.32	1.29	1.30
Dual Quaternions DLB	1.02	1.01	1.01	1.03	1.01	1.02
Dual Quaternions Sclerp	2.04	2.04	2.04	2.09	2.04	2.05

Table 3.9: Running Time for Implementation of Interpolation Methods

We also looked at a test case, involving the transformations,

$$\begin{aligned} &Angle = 30^\circ, && Axis = (0,1,0), && Translation = (2,0.5,1) \\ &Angle = 170^\circ, && Axis = (0.707,-0.707,0), && Translation = (10,-1,-4) \end{aligned}$$

In Figure 3.4 and Figure 3.5 we see the results of the interpolation between these two transformations, as projected onto the $(1,0,0)$ and $(0,1,0)$ axis. These diagrams show the effects of the resulting interpolation on both the vector $(0,0,0)$ and the vector $(1,1,1)$, showing just the

translation and the translation and rotation. We can see how the dual quaternion interpolation methods do not just rotate around the origin, but choose an ideal centre for interpolation. We can also see the difference in the Sclerp and Nlerp methods, and the undesirable effects that occur with matrix interpolation.

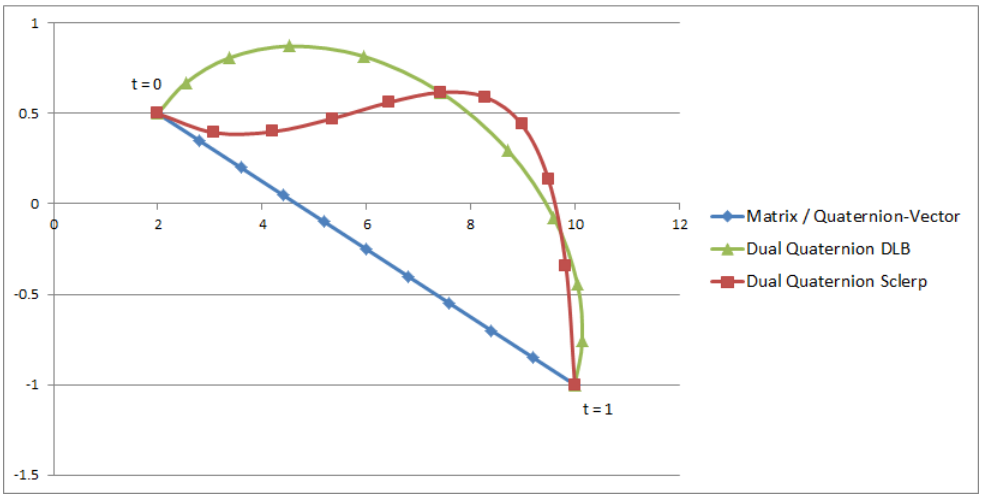


Figure 3.4: Interpolated transformation translation component

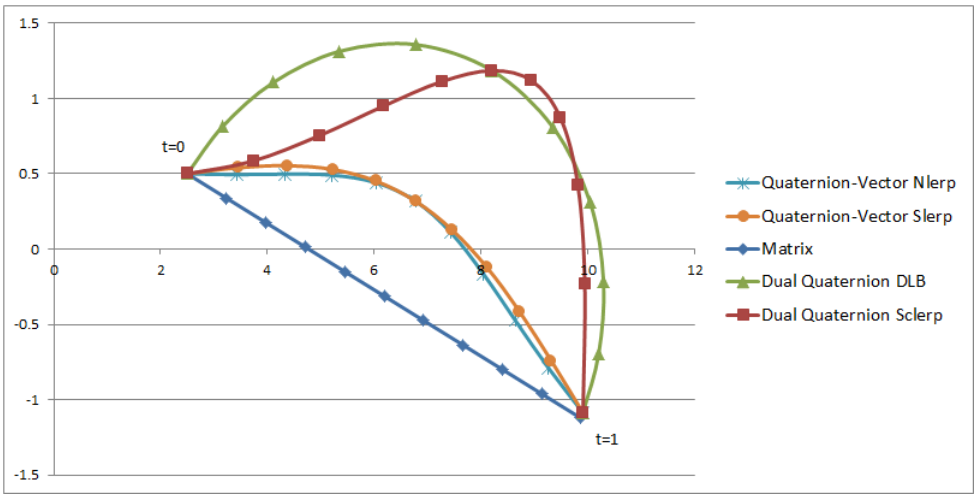


Figure 3.5: Modification of point $(1, 1, 1)$ by interpolated transformation

In Figure 3.6, Figure 3.7 and Figure 3.8 we can see the results applied to a transformation hierarchy. To show only the effects of the transformation, the skinning does not blend vertices, that is each vertex only corresponds to only one transformation. We can see that in this case the quaternion-vector and dual quaternion are near identical, but the matrix is no longer rigidly transforming the vertices, and is scaling them down.

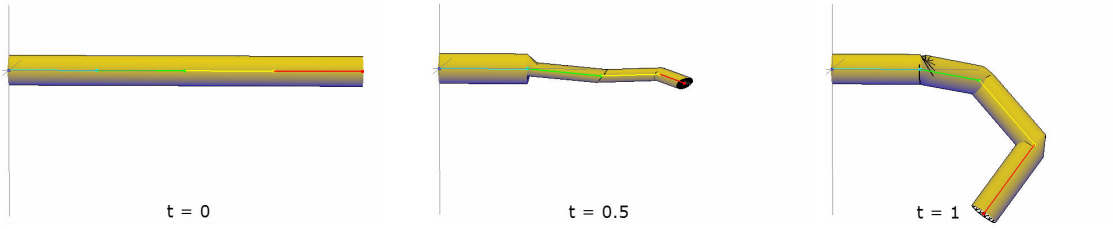


Figure 3.6: Matrix Hierarchical Interpolation

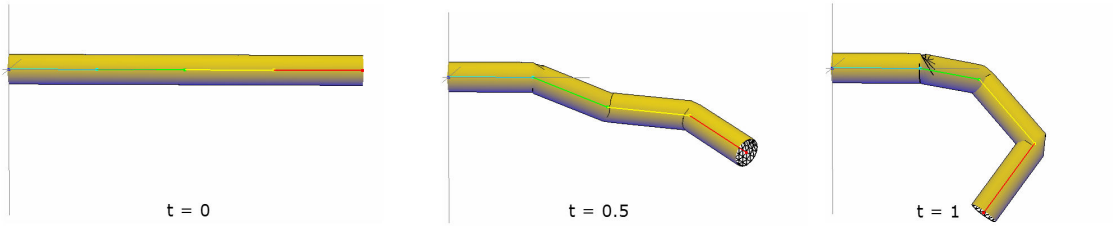


Figure 3.7: Quaternion-Vector Hierarchical Interpolation

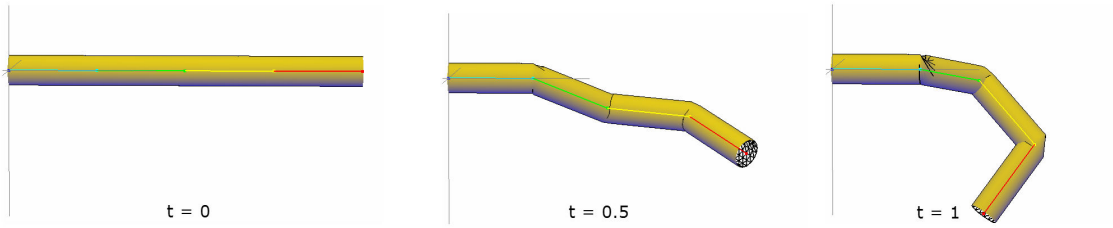


Figure 3.8: Dual Quaternion Hierarchical Interpolation

3.6 Geometric Skinning

3.6.1 Method

The final properties of rigid transformations we investigated were the blending properties of the rigid transformation methods, particularly in relation to geometric skinning. Blending is similar to interpolation, but instead of trying to construct a smooth path between two transformations, instead we have any number of transformations, with weights assigned to each. Since we already have compared the interpolation properties, we looked at the skinning application to understand how the blending performs, and examine its potential applications.

Three blending methods were compared, chosen based on being the most common methods used for the given transformation types. Linear blending was analysed for matrices, DLB was

analysed for dual quaternions and direct quaternion blending was analysed for quaternion-vector pairs. Spherical blend skinning was also researched but not implemented.

The first experiment compared the efficiency of the methods for skinning by looking at the blending time plus the time to modify vertices. For the other experiment, we used the fully functioning skeletal system. A basic testing mesh was constructed, which consisted of a number of links like those in Figure 3.9, with accompaning bones and weights. This way the basic blending properties could be assessed, and the skinning methods compared.

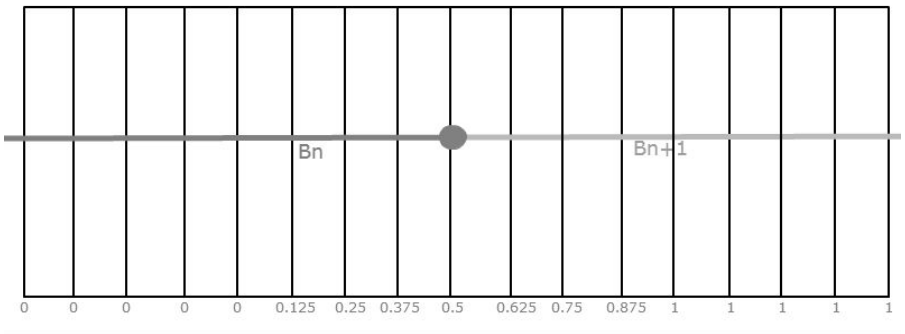


Figure 3.9: Mesh used for skinning (with linear weighting)

3.6.2 Mathematical Analysis

DLB is simply extended from interpolation methods to blend any number of transformations as

$$\mathrm{DLB}(d_1, w_1, d_2, w_2, \ldots, d_n w_n) = \frac{d_1 w_1 + d_2 w_2 + \cdots + d_n w_n}{||d_1 w_1 + d_2 w_2 + \cdots + d_n w_n||},$$

DQB is extended similarly from Nlerp for quaternion-vector. Sclerp and Slerp are more difficult, and require slow iterative methods in order to find results[19]. In addition, due to the noncommutative nature of Slerp[3], the result will vary based on the order of the operands.

When skinning, the change in rotation centres is very important, since when blending between bone transformations, each bone is supposed to rotate around the centre of the parents. Thus the coordinate dependence will change the way the bones interpolate. Spherical blend skinning is the method Kavan and Zara present[16] by which the ideal rotation centre is found. The method finds the point at which modification by all of the transformations, produces the smallest translation. Finding this involves finding a least squares solution for p to all the equations

$$r_a * p = r_b * p$$

given by every possible combination of rigid transformation r_a and r_b from the transformations to be blended. Due to this being a slow iterative process, Spherical Blend Skinning is itself a slow operation.

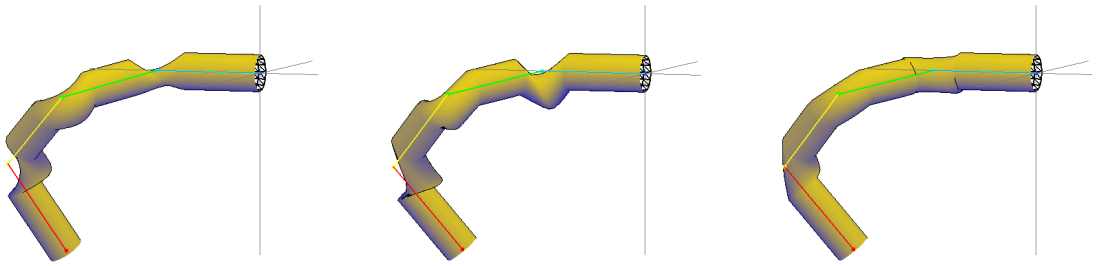
3.6.3 Results

The running time of the skinning is shown below in Table 3.10. This shows the amount of time to blend three transformations, and apply that transformation to a point. While the results show that the direct quaternion blending is faster than other methods, it should be noted again that the matrices are not implemented as they would normally be optimised, and so this will have some bearing on these results.

	Test 1	Test 2	Test 3	Test 4	Test 5	Mean
Matrix Linear Blending	4.51	4.51	4.51	4.50	4.52	4.51
Direct Quaternion Blending	1.47	1.47	1.47	1.47	1.48	1.47
Dual Quaternion Linear Blending	2.17	2.19	2.17	2.18	2.18	2.18

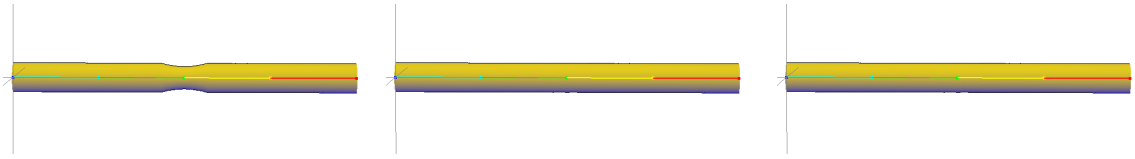
Table 3.10: Running Time for Modifying Points with Transformations

Results showing the blending for skinning are shown in Figure 3.10 and Figure 3.11. Figure 3.10(a) shows how the scaling of linear matrix blending affects the joints. Figure 3.10(b) shows how the rotation around the origin affects the blending, and Figure 3.10(c) shows how DLB blends naturally due to coordinate independant blending. Figure 3.11 shows a transformation that is simply a rotation around the same axis that the cylinder lies. Figure 3.11(a) shows how the matrices have a effect known as the ”candy-wrapper” effect caused by the blending not producing a rigid transformation.



(a) Matrix Linear Blending (b) Direct Quaternion Blending (c) DLB

Figure 3.10: Comparison of blending methods A



(a) Matrix Linear Blending (b) Direct Quaternion Blending (c) DLB

Figure 3.11: Comparison of blending methods B

4 Discussion

Looking at the current results it would seem that there is no particular reason to use matrices, other than their simplicity, and proliferation through graphics APIs. However, while we did perform a number of tests, the efficiency tests were likely wrong in the case of matrices. Particularly comparing this to the ideal number of operations used for matrix multiplication, the results are significantly slower. In fact, the speed of modifying points should be faster for matrices given the calculated number of operations. However while extra work should be done in the future to get better results, this does highlight an important point: the implementation can matter more than the actual representation used. Using a matrix to modify vertices may be faster than a dual quaternion but using a poor implementation is, as was in fact shown, worse. Likewise a poor quaternion implementation would have made the quaternion types slower.

The skeletal system, in particular the skinning, shows off the problems with using matrices and the effectiveness of dual quaternions for blending and interpolation. The skinning results showed us that dual quaternions avoid many of the issues that matrices have, and perform better than vector quaternions, due to the blending occurring around an ideal rotation centre. Because of this they avoided the candy wrapper effect, and produced a smooth interpolation, while being just as efficient. Using Nlerp for dual quaternions was very efficient, and given the rotational difference, would be ideally preferred over Slerp for skinning in real time applications. In the case of skeletal hierarchies, using Slerp may be a better idea since the errors may combine and for especially long chains could add up to a large interpolation error using Nlerp.

Interestingly, for quaternion-vector pairs, the rotation centre being the origin can be useful in certain cases. For considering a bone hierarchy, when the translational component is identical, the interpolation produces the same result. Most skeletal animation will only have the rotational component of the bones changing so the interpolation then only needs to be performed on the rotational component. Also, in this case the origin is the position of the the previous bone, so interpolating with the origin as the centre makes sense. However because the skinning methods for these pairs do not well, they are not ideal for a full skeletal animation system since either the skinning would be worse, or the time taken to generate the rotation centers would negate the benefit.

These blending properties also allow for smooth spline based interpolation to be performed. Because Nlerp and DLB functions similarly to a linear combination of vectors in 7 and 8 dimensional space respectively, the constants can be calculated for a polynomial or spline connecting points together. These constants can then be used to blend the points by using them as weights to

the blending function. Thus one can create a path through the transformation space that should be smooth. In the case of dual quaternions, the coordinate invariance of the interpolation is a desirable property since the path will not change with a transformation to a different space.

We found that dual quaternions do not have the speed or simplicity of quaternion-vector pairs or rigid transformation matrices when it comes to basic combination or modification using the transformations. Matrices have a simple representation, are straightforward to multiply and can be fast at transforming points. Quaternion-vector pairs are a simple extension of rotation quaternions and have more efficient and more accurate results than dual quaternions. Thus when the representations needed only requires combinations of transformations or simple modification of points, dual quaternions are likely not the best representation.

However, when it comes to performing interpolation and blending, dual quaternions have several desirable properties over other representations. Due to the quaternion nature, the interpolation is shortest path, and dual quaternions rotate around ideal centres. While DLB is more inaccurate than Sclerp, it maintains the ideal rotational interpolation properties while being of a similar speed to other methods. Also, while dual quaternions are slower in most cases than the linear interpolation methods on other representations, the speed is still much less than using complicated methods to achieve the same result. Thus when the interpolation needs to be coordinate invariant and rotate around ideal centres, as well as maintaining shortest path and potentially constant speed, dual quaternions are the best choice.

4.1 Future Work

Due to the issues with matrices being a lot slower than the theoretical results would suggest, better implementations should be looked into to produce better results. This was perhaps an area of the study that was the weakest, so future studies into the effectiveness of dual quaternions should consider it. In general more results should be gathered for all the areas of study , particulary for the interpolation methods, to get a more accurate picture of the application.

Results for the slower methods such as spherical blend skinning and skinning with log matrix blending would allow better comparison with all the methods. This would help to get a better understanding of the usefulness of dual quaternions. The skeletal system could also benefit from more features. Inverse kinematics (IK) in particular would be an interesting application of dual quaternions, in comparison to existing IK methods.

There are additional areas of study that would also be of interest for further study. It would be of particular interest to look at rigid transformations for rigid coordinate changes and how those can be solved and combined, and also how useful the representations are in calculations of this kind. Their use in blending motion could be further investigated since while there are some similarites to skeletal animation, it would provide a better understanding of when the blending and interpolation methods are useful.

5 Conclusions

Our goal for this project was to compare the different rigid transformation methods, gain an understanding of why they are like this and understand the potential applications of dual quaternions. We did this by looking at previous research on dual quaternions and rigid transformations, gaining a mathematical understanding of how they worked, and constructing the skeletal animation system to compare and test results.

Matrices we found to be ineffective at interpolation and blending, producing non rigid transformations with simplistic algorithms or requiring complex methods to interpolate rigidly or with shortest path. Their simplicity of implementation and speed (in certain cases) does provide some benefit when rigid transformation interpolation is not required. More work would need to be done on making the implementations better so that a fairer comparison between the methods could be made. Using quaternion-vector pairs was shown to solve the interpolation issues with little difference in speed, accuracy and with maintaining some of the simplicity of matrices.

Dual Quaternions provided similar improvements to the pairs over matrices in the areas of blending and interpolation. As we saw from the skeleton interpolation, the quaternion-vector pairs interpolated the rotation component the same. The only difference was the change in translation, caused by a the interpolating happening around a different centre of rotation. Unlike quaternion-vector pairs, dual quaternions are coordinate invariant, so the interpolation will be the same no matter the coordinate space the rotation is performed.

The skeleton hierarchical interpolation demonstrated that the quaternion based interpolation is desirable as it is shortest path and interpolates rigidly. The skinning showed us that the coordinate invariant blending properties of dual quaternions made them. Despite the extra computation and memory required from using quaternions and vectors, we found that dual quaternions are a natural fit for skeletal animation systems.

Given the special properties that dual quaternions have when representing rigid transformations, there may be other applications where dual quaternion perform better than other representations. Using them when considering the motion of rigid bodies would be an interesting area of study to look into. Given that this requires many of the same properties as skeletal animation, it is likely dual quaternions are also a better fit there than other representation methods.

Bibliography

[1] J Harold Ahlberg, Edwin Norman Nilson, and Joseph Leonard Walsh. The theory of splines and their applications. *Mathematics in Science and Engineering, New York: Academic Press, 1967*, 1, 1967.

[2] Marc Alexa. Linear combination of transformations. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 380–387. ACM, 2002.

[3] Charles Bloom, J Blow, and C Muratori. Errors and omissions in marc alexas linear combination of transformations, 2004.

[4] Jonathan Blow. Understanding slerp, then not using it. *Game Developer Magazine*, 2004.

[5] Nester Burtnyk and Marceli Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM*, 19(10):564–569, 1976.

[6] Edwin Catmull and Raphael Rom. A class of local interpolating splines. *Computer aided geometric design*, 74:317–326, 1974.

[7] JJ Cervantes-Sánchez, JM Rico-Martínez, G González-Montiel, and EJ González-Galván. The differential calculus of screws: Theory, geometrical interpretation, and applications. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 223(6):1449–1468, 2009.

[8] William K Clifford. Preliminary sketch of bi-quaternions. *Proc. London Math. Soc*, 4 (381-395):157, 1873.

[9] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation*. Datalogisk Institut, Københavns Universitet, 1998.

[10] JR Dooley and J Michael McCarthy. Spatial rigid body dynamics using dual quaternion components. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 90–95. IEEE, 1991.

[11] David W Eggert, Adele Lorusso, and Robert B Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5-6): 272–290, 1997.

[12] Sven Forstmann, Jun Ohya, Artus Krohn-Grimberghe, and Ryan McDougall. Deformation styles for spline-based skeletal animation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–150. Eurographics Association, 2007.

[13] Ivo Zoltan Frey and Ivo Herzeg. Spherical skinning with dual quaternions and qtangents. In *SIGGRAPH Talks*, page 11, 2011.

[14] QJ Ge, Amitabh Varshney, Jai P Menon, and Chu-Fei Chang. Double quaternions for motion interpolation. In *Proceedings of the ASME Design Engineering Technical Conference*, 1998.

[15] William Rowan Hamilton. Ii. on quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 25 (163):10–13, 1844.

[16] Ladislav Kavan and Jiří Žára. Spherical blend skinning: a real-time deformation of articulated models. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 9–16. ACM, 2005.

[17] Ladislav Kavan, Steven Collins, Carol OSullivan, and Jiri Zara. Dual quaternions for rigid transformation blending. *Technical report*, 2006.

[18] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46. ACM, 2007.

[19] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics (TOG)*, 27(4):105, 2008.

[20] Ben Kenwright. A beginners guide to dual-quaternions. .

[21] Ben Kenwright. Dual-quaternions. .

[22] J Michael McCarthy. *Introduction to theoretical kinematics*. MIT press, 1990.

[23] Ramakrishnan Mukundan. *Advanced Methods in Computer Graphics: With Examples in OpenGL*. Springer, 2012.

[24] Ken Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH computer graphics*, 19(3):245–254, 1985.

[25] Ken Shoemake and Tom Duff. Matrix animation and polar decomposition. In *Proceedings of the conference on Graphics interface*, volume 92, pages 258–264. Citeseer, 1992.

[26] Christopher Twigg. Catmull-rom splines. *Computer*, 41(6):4–6, 2003.

[27] An T Yang and Ferdinand Freudenstein. Application of dual-number quaternion algebra to the analysis of spatial mechanisms. *Journal of Applied Mechanics*, 31:300, 1964.